Journal of Automata, Languages and Combinatorics **u** (v) w, x–y

The ld and dlad Bio-Operations on Formal Languages [1]

MARK DALEY
Department of Computer Science, University of Saskatchewan
Saskatoon, Saskatchewan, S7N 5A9, Canada
e-mail: daley@cs.usask.ca

OSCAR H. IBARRA
Department of Computer Science, University of California
Santa Barbara, CA 93106, USA
e-mail: ibarra@cs.ucsb.edu

LILA KARI, IAN MCQUILLAN
Department of Computer Science, University of Western Ontario
London, ON N6A 5B7, Canada
e-mail: {kari,imcquill}@csd.uwo.ca

and

KOJI NAKANO
School of Artificial Complex Systems Engineering, Hiroshima University
Kagamiyama, Higashi-Hiroshima 739-8527, Japan
e-mail: knakano@fse.hiroshima-u.ac.jp

ABSTRACT

We continue the language theoretic study of operations suggested by the gene unscrambling process in stichotrichous ciliates. One of the two complementary models of gene unscrambling is based on operations inspired by the ways in which a DNA molecule can fold: hi (hairpin loop with inverted pointers) which reverses a substring between a pointer sequence and its reverse, ld(loop with direct pointers)-excision which deletes a substring between two pointers and dlad(double loop with alternating direct pointers)-excision / reinsertion which swaps two substrings marked by pointer-pairs. We specifically consider the closure properties of several families of languages under the operations ld and $dlad$ and the solvability of language equations involving these operations.

Keywords: Theoretical DNA computing, bio-operations, closure properties, formal languages, decision questions

[1]Research of Mark Daley and Lila Kari has been supported by Natural Sciences and Engineering Council of Canada Grants. Research of Oscar H. Ibarra and Koji Nakano has been supported by the Japan Society for the Promotion of Science (JSPS) Research Program S02251; Oscar H. Ibarra was also supported by NSF Grants IIS-0101134 and CCR02-08595.

1. Introduction

The stichotrichous ciliates, a class of single-celled organisms, has recently been the motivation for a number of papers in formal language theory. One of the intriguing properties of ciliates is that they keep their genetic material in a scrambled form and thus need some form of computational apparatus in order to descramble their genes and produce functional proteins. For example, the gene encoding *DNA polymerase* α in the ciliate *Stylonichia Lemnae* is broken into more than 48 discrete pieces that must be reassembled in the correct order to produce a functional gene [12]. The biological details of this process are still not yet fully understood but are currently the subject of active investigation. Further details on the biology of this process can be found in [15, 16, 17].

The formal study of this process has lead us not only to a better understanding of ciliate genetics, but has also produced two models of computation that are inherently interesting in their own right.

The first model, proposed by Kari and Landweber [10, 11] proposes two atomic operations based on circular insertions and deletions guided by pointers within the gene. The second model, motivated by Ehrenfeucht, Harju, Petre, Prescott and Rozenberg [4, 5, 18], consists of three operations inspired by the ways in which a DNA molecule can fold: hi (hairpin loop with inverted pointers) which reverses a substring between a pointer sequence and its reverse, ld(loop with direct pointers)-excision which deletes a substring between two pointers and $dlad$(double loop with alternating direct pointers)-excision / reinsertion which swaps two substrings marked by pointer-pairs.

While the full details of the biological mechanisms underlying this process are not yet completely understood, the reader may find further information in [15, 16, 17].

In this paper we will consider the properties of the ld and $dlad$ operations from an abstract formal-language-theoretic viewpoint and present some new results on the hi operation as well. For similar studies of the hi operator and the operations in the Kari-Landweber model, the reader is referred to [2, 3].

The layout of the paper is as follows: in Section 2 we define the ld and $dlad$ bio-operations and investigate the closure properties of NCM and its generalizations. NCM is the class of languages defined by nondeterministic finite automata augmented with reversal-bounded counters (i.e., the counters can be incremented/decremented by 1 and tested for zero but the number of alternations between nondecreasing mode and nonincreasing mode and vice-versa is bounded by a fixed constant) [8]. It is known [6] that NCM is the smallest class of languages containing the regular sets that is closed under the operations of homomorphism, intersection, and shuffle (section 2 gives the precise definition). Section 3 considers the closure properties of languages defined by time- and space-bounded Turing machines. Section 4 investigates the closure properties of two families of L-systems, namely 0L and ET0L under hi and $dlad$. Section 5 briefly looks at language equations involving the $dlad$ and ld operation and provides some general results concerning any language equation involving a unary operator. Section 6 provides a summary and conclusions.

The notations used in the paper are summarized as follows. An alphabet Σ is a finite nonempty set. A word w over Σ is an element of the free semigroup (denoted Σ^+) generated by the letters of Σ and the catenation operation. The length of a word, written $|w|$, is equal to the number of letters in the word. In the free monoid Σ^* we also allow the empty word λ

where $|\lambda| = 0$. A language L is a, possibly infinite, set of words over a given alphabet. The complement of a language L is written L^c and is defined as $L^c = \Sigma^* \setminus L$.

For further details on basic formal language theory, the reader is referred to [19].

2. The Families NCM, NFCM, and NPCM

In this section, we investigate the closure properties of various families (or classes) of languages under ld and $dlad$ bio-operations [5, 18, 4]. Our objective is to identify large classes of recursive languages that are closed under these bio-operations. We will use the terms "family" and "class" interchangeably in the paper. We first define the ld operation from [4].

Definition 1 *Let α be a word in Σ^+. The ld operation on α, denoted by ld(α) is defined as* ld$(\alpha) = \{xpz | \alpha = xpypz, x, z \in \Sigma^*, y \in \Sigma^+, p \in \Sigma^+\}$.

In the definition above, we say p is a pointer. The definition can then be extended to languages:

$$\text{ld}(L) = \bigcup_{\alpha \in L} \text{ld}(\alpha), \text{ for any } L \subseteq \Sigma^+.$$

We now show that allowing pointers of arbitrary length is equivalent to allowing pointers of length one.

Lemma 1 *For any $w \in \Sigma^+$, $ld(w) = \{xaz | w = xayaz, x, z \in \Sigma^*, y \in \Sigma^+, a \in \Sigma\}$.*

Proof. "$\subseteq$" Consider $u \in ld(w)$. Then u is of the form xpz where $x, z \in \Sigma^*$ and $p \in \Sigma^+$. We can thus write $u = xp_1p_2...p_nz$, $n \geq 1$ where $w = xp_1p_2...p_nyp_1p_2...p_nz$ and $p = p_1p_2...p_n$ with $p_i \in \Sigma$. We now rewrite $u = (xp_1p_2...)p_nz = x'p_nz$ and $w = (xp_1p_2...)p_n(yp_1p_2...)p_nz = x'p_ny'p_nz$ where $x' = xp_1p_2...p_{n-1}$ and $y' = yp_1p_2...p_{n-1}$ and the inclusion follows.

"$\supseteq$" Obvious. □

Next, we give the definition of the $dlad$ operation [4].

Definition 2 *Let α be a word in Σ^+. The $dlad$ operation on α, denoted by dlad(α) is defined as* dlad$(\alpha) = \{upbqwpaqv | \alpha = upaqwpbqv, u, w, v \in \Sigma^*, a, b, p, q \in \Sigma^+\}$.

We say that p and q are pointers above. This definition is extended to languages as follows:

$$\text{dlad}(L) = \bigcup_{\alpha \in L} \text{dlad}(\alpha), \text{ for any } L \subseteq \Sigma^+.$$

Similarly to the result for ld above, we now show that we can consider pointers of length one without loss of generality in the $dlad$ operation.

Lemma 2 $dlad(w) = \{xa\beta bya\alpha bz | w = xa\alpha bya\beta bz, x, y, z \in \Sigma^*, \alpha, \beta \in \Sigma^+, a, b \in \Sigma\}$.

Proof. "$\subseteq$" Consider $u \in dlad(w)$. Then u is of the form $u = xp\beta qyp\alpha qz$ where $w = xp\alpha qyp\beta qz$ and $p, q, \alpha, \beta \in \Sigma^+$, $x, y, z \in \Sigma^*$. Now let $p = p_1p_2...p_n$, $q = q_1q_2...q_m$ where $p_i, q_i \in \Sigma$, $n, m \geq 1$. We can now write $u = xp_1p_2...p_n\beta q_1q_2...q_myp_1p_2...\ p_n\alpha q_1q_2...q_mz$ and $w = xp_1p_2...p_n\alpha q_1q_2...q_myp_1p_2...p_n\beta q_1q_2...q_mz$. Take $x' = x$, $a = p_1$, $b = q_1$, $\alpha' = p_2...p_n\alpha$, $y' = q_2...q_my$, $\beta' = p_2...p_n\beta$ and $z' = q_2...q_mz$ then $u = x'a\beta'by'a\alpha'bz'$ where $w = x'a\alpha'by'a\beta'bz'$ and the inclusion follows.

"$\supseteq$" Trivial. □

For $k \geq 0$, let NCM(k) be the class of nondeterministic one-way finite automata augmented with k reversal-bounded counters, and NCM be the union of such classes over all k's (see [8] for details). Thus, at every step, a counter can be incremented by 1, decremented by 1, or not changed, and it can be tested for zero. It is reversal-bounded in that in any computation, the number of alternations between nondecreasing mode and nonincreasing mode and vice-versa is bounded by a given constant. For notational convenience we also use NCM(k) and NCM to denote the respective families of accepted languages, and use the same notation to refer to a machine in the classes. Clearly, NCM defines a large class of languages (all regular sets, some non-context-free languages, etc.) Note than an NCM(0) has *no* counter and is an ordinary finite automaton. Hence, the class NCM(0) = REG = regular sets.

Let NPCM be an NCM augmented with an unrestricted pushdown stack. An NFCM is an NCM augmented with a free (i.e., unrestricted) counter. An NPCM (NFCM) with k reversal-bounded counters will be denoted by NPCM(k) (NFCM(k)). Thus, an NPCM(0) (NFCM(0)) is just an ordinary pushdown (one-counter) automaton. Hence, NPCM(0) = CF = context-free languages, and NFCM(0) = one-counter languages.

There is a nice characterization of NCM (respectively, NPCM, NFCM) in terms of regular sets (respectively, context-free languages, one-counter languages). The *shuffle* $u \amalg v$ of two words $u, v \in \Sigma^*$ is a finite set consisting of the words $u_1v_1 \ldots u_kv_k$, where $u = u_1u_2 \ldots u_k$ and $v = v_1v_2 \ldots v_k$ for some $u_i, v_i \in \Sigma^*$. If L_1 and L_2 are two languages, their *shuffle* is the language

$$L_1 \amalg L_2 = \bigcup_{u \in L_1, v \in L_2} u \amalg v.$$

A *simple shuffle language* is a language of the form $\Sigma^* \amalg \{d^ne^n \mid n \geq 0\}$, for some alphabet Σ and distinct symbols d, e.

It is known [6] that NCM (respectively, NPCM, NFCM) is the smallest class of languages containing the regular sets (respectively, context-free languages, one-counter languages) that is closed under homomorphism and intersection with simple shuffle languages. In particular, since NCM is clearly closed under homomorphism, intersection, and shuffle (i,e., if L_1 and L_2 are in NCM, then $L_1 \amalg L_2$ is also in NCM), we see that NCM is the smallest class of languages containing the regular sets that is closed under homomorphism, intersection, and shuffle.

We now proceed to examine the closure of NCM (respectively, NPCM, NFCM) under ld and $dlad$ operations. We will need the following proposition which is easily verified using standard constructions:

A *full trio* is any family of languages closed homomorphism, inverse homomorphism, and intersection with regular sets.

Proposition 1 *NCM(k), NPCM(k), and NFCM(k) are closed under homomorphism, inverse homomorphism, and intersection with regular sets.*

Proposition 2 *Every full trio is closed under ld.*

Proof. Let L be a language over the alphabet Σ. For every symbol a in Σ, let $a.1, a.2, a.3$ be new distinct symbols, and $\Sigma_1, \Sigma_2, \Sigma_3$ be the set of all such symbols, respectively. We can think of $a.1, a.2, a.3$ as "marked" versions of symbol a.

Define a homomorphism h from $(\Sigma \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma_3)^*$ to Σ^* as follows: $h(a) = h(a.1) = h(a.2) = h(a.3) = a$ for all a in Σ. Let $L' = h^{-1}(L) \cap \{\Sigma^* a.1 \Sigma_2^+ a.3 \Sigma^* \mid a \in \Sigma\}$ Now define another homomorphism g as follows: For all a in Σ, $g(a) = g(a.3) = a$ and $g(a.1) = g(a.2) = \lambda$. Clearly, $ld(L) = g(L')$. □

The next result now follows from the two propositions above.

Corollary 1 *NCM(k), NPCM(k), and NFCM(k) are closed under ld.*

Note that in the above proof, the purpose of the new symbols and the inverse homomorphism and intersection with regular sets is to obtain from L a language L' where the different components (segments) of the string are marked, i.e., every string in L' is of the form $xpypz$, where the string x, the first p (which is a symbol), the string y, the second p, and the string z use different alphabets. In this section, when we say that the components of a string are marked, we will mean that they use different alphabets, i.e., inverse homomorphism and intersection with regular set have already been done. Also, for notational convenience, we will also refer to language L' and the machine M' accepting it simply as L and M, respectively. Applying the homomorphism to remove the marks is also straightforward and we will assume this is done explicitly at the end of the construction in the proofs. The next proposition concerns the $dlad$ operation.

Proposition 3 *NFCM and NCM are closed under $dlad$.*

Proof. Given an NFCM M accepting L, we will construct another NFCM M' accepting $dlad(L)$.

A configuration of M is a triple $\alpha = (q, f, R)$, where q is the state, f is the value of the free counter, and R is an array of values of the reversal-bounded counters. When we say that M' "records" a configuration α, we mean, it stores the state in its finite control, and stores the values of the free counter and reversal-bounded counters into an auxiliary set of counters. By using additional counters, we may assume that the original values of the free counter and reversal-bounded counters are preserved. Thus, after recording α, M' can still continue its computation (from configuration α) using the original counters. We now describe the computation of M', given a (marked) input of the form: $vpyqxpwqz$.

M' will simulate the computation of M on $vpwqxpyqz$. M' first simulates M on input segment vp. After processing p, M will be in some configuration α_p. Without moving its input head, M' guesses a configuration α_w that M would enter starting on α_p on some input w. It records (α_p, α_w). M' also guesses two configurations β_p and β_y and records two copies of each of these configuration using auxiliary counters. M' nows moves its input head

and simulates the computation of M on y starting in configuration β_p. After processing y, M checks that the configuration reached is β_y. M' then continues the simulation of M on input segment qxp starting in configuration α_w. After processing qxp, M' checks that M is in configuration β_p. Then reading y, M' checks that M when started in α_p indeed enters configuration α_w. Finally, M' completes the simulation of M on the remaining input segment qz starting in configuration β_y and accepts if M accepts. It is easily verified that M' accepts $dlad(L(M))$. Note that when M is an NCM (i.e., it has no free counter), then M' is also an NCM. □

Clearly, in the construction above, when M does not have any counter, then M' also does not need any counter. Moreover, as mentioned earlier, the family of regular languages are exactly the languages accepted by finite acceptors not augmented by any storage (and are consequently equal to NCM(0)). Hence:

Corollary 2 *The family of regular languages is closed under $dlad$.*

The family of context-free languages are equal to the languages accepted by finite acceptors augmented by exactly one pushdown (and thus are equal to NPCM(0)). Similarly, the family of one-counter languages are the languages accepted by acceptors with exactly one free counter (and are equal to NFCM(0)).

Proposition 4 *The family of context-free languages and the family of one-counter languages are not closed under $dlad$:*

Proof. Let a, b, c, d, p, q be distinct symbols, and $L = \{a^n p b^n q p c^m q d^m | n, m \geq 1\}$. Clearly, L is NFCM(0) and, hence, also in NPCM(0). But $dlad(L) = \{a^n p c^m q p b^n q d^m \mid n, m \geq 1\}$ is not context-free. □

Remark: We do not believe that Proposition 3 can be generalized to hold for NPCM. Let a, b, c, d, p, q be distinct symbols, and $L = \{xpx^r qpyqy^r \mid x \in \{a, b\}^*, y \in \{c, d\}^*\}$. L is a context-free language, hence in NPCM. However, we believe that $dlad(L) = \{xpyqpx^r qy^r \mid x \in \{a, b\}^*, y \in \{c, d\}^*\}$ is not in NPCM.

The construction in Proposition 3 can be generalized in various ways. For example, let ϕ be a permutation of $(1, ..., k)$. For a word α in Σ^+, and a permutation ϕ, define $\phi(\alpha) = \{wpx_{\phi(1)}qy_1...px_{\phi(k)}qy_k \mid \alpha = wpx_1qy_1...px_kqy_k,\ w \in \Sigma^*, x_1, y_1, ...,\ x_k, y_k \in \Sigma^+, p, q \in \Sigma^+\}$. For a language L, $\phi(L)$ is defined in the obvious way. One can show that NFCM is closed under ϕ. In fact, one can also have a second permutation ϕ' and apply this simultaneously on $y_1, ..., y_k$, and the closure still holds.

We can also specify that some of the x_i's (and y_i's) are to be "reversed", e.g., x_2 is replaced by x_5^r (which is the reverse of x_5) and x_3 is replaced by x_1^r. Then the result would still be valid. For example, suppose we modify the definition of $dlad$ to $r - dlad$ as follows:

Definition 3 *For α in Σ^+, define r-dlad$(\alpha) = \{vpy^r qxpwqz | \alpha = vpwqxpyqz\}$.*

We will show that if L is in NFCM (respectively, NCM), then $r-dlad(L)$ is also in NFCM (respectively, NCM). We will need the following result in [2, 8]:

Proposition 5 *NCM and NFCM are closed under reversal.*

Lemma 3 *If M is an NFCM (NCM), define $L_c = \{\alpha\beta y \mid \alpha, \beta$ are configurations of M, and M on input y starting in configuration α reaches configuration $\beta\}$. (Assume that the states and counter values in configurations α and β are written in unary using distinct symbols.) Then*

1. *L_c can be accepted by an NFCM (NCM) M_c.*
2. *L_c^r (the reverse of L_c) can be accepted by an NFCM (NCM) M_c^r.*

Proof. The construction of M_c is straightforward. M_c reads α and configures the free counter and reversal-bounded counters to the values specified in α. Then it reads β and records this configuration using another set of reversal-bounded counters. M_c then simulates the computation of M on input y, and checks that the final configuration is β. The second part follows from the above proposition. □

Note that in the definition of L_c above, the placement of α and β in the string is not important, i.e., they can appear before y, β can appear before α, or y can appear between the two configurations. This is because M_c can use several sets of auxiliary counters, and before simulating M, M_c can guess the values in α and make two sets of copies, simulates M on input y using one set of copy, and later when it sees α on the input, checks that the values in α agree with those that were recorded. The configurations could also be written in reverse, i.e., α^r and β^r.

Proposition 6 *NFCM and NCM are closed under $r - dlad$.*

Proof. The proof is a modification of the construction in the proof of Proposition 3, using M_c^r.

M', when given a (marked) input $vpy^rqxpwqz$, simulates the computation of M on $vpwqxpyqz$. M' first simulates M on input segment vp. After processing p, M will be in some configuration α_p. Without moving its input head, M' guesses a configuration α_w that M would enter starting on α_p on some input w. It records (α_p, α_w). M' also guesses two configurations β_p and β_y and records two copies of each, and then moves its its input head on y^r and simulates the computation of M_c^r on $y^r\beta_y^r\beta_p^r$. Since β_y^r and β_p^r are not present in the input, M' uses the values recorded in the auxiliary counters (as part of the input) in the simulation of M_c^r. M' then continues the simulation of M on input segment qxp. The rest of the construction is the same as in Proposition 3. □

Similarly, we can define r-dlad$(\alpha) = \{vpy^rqxpw^rqz | \alpha = vpwqxpyqz\}$, and the proposition above still holds.

Finally, consider the model of a two-way nondeterministic finite automaton (with endmarkers) augmented with finitely many reversal-bounded counters (2NCM). Such a machine is *finite-crossing* if there is a constant k such that in any computation, the input head crosses the boundary between any two adjacent cells of the input at most k times. It is known that any finite-crossing 2NCM can be converted to an equivalent NCM [8]. Hence, all the results above are valid for finite-crossing 2NCMs.

3. Space-Bounded and Time-Bounded Turing Machines

In this section, we investigate the closure properties of space-bounded and time-bounded Turing Machine (TM) complexity classes under ld and dlad.

For a space bound $S(n)$, let *NSPACE*$(S(n))$ be the class of languages accepted by $S(n)$ space-bounded nondeterministic Turing machines, and *DSPACE*$(S(n))$ be the deterministic class. Thus, these machines have a two-way read-only input tape (with endmarkers) and multiple read-write worktapes which are $S(n)$ space-bounded. (It is known that any number of worktapes can be merged into one worktape with the same space bound.) Throughout, we assume that $S(n) \geq log\ n$. Note that *NSPACE*(n) and *DSPACE*(n) are the classes of context-sensitive and deterministic context-sensitive languages, respectively.

Proposition 7 *NSPACE(*$S(n)$*) and DSPACE(*$S(n)$*) are closed under* $dlad$.

Proof. First consider the case *NSPACE*$(S(n))$. Let M be an nondeterministic TM with a two-way read-only input (with endmarkers) and an $S(n)$ space-bounded read-write worktape. We construct an $S(n)$ space-bounded nondeterministic TM M' accepting $dlad(L(M))$. Without loss of generality, we may construct M' such that it hasseveral $S(n)$ space-bounded read-write worktapes (since any number of worktapes can be easily merged into one).

Given an input of the form $vpyqxpwqz$, M' will simulate the computation of M on $vpwqxpyqz$ by first nondeterministically choosing the locations of the pointers p, q, p, q, and recording them in binary on the worktape using $log\ n$ space. Then the simulation of M is implemented by M' in the obvious way: the simulation on the input segments vp, qxp, and qz is obvious. When M computes on input segment w and y, M' moves its input head to the appropriate pointer locations (stored in the worktapes).

For the case of *DSPACE(*$S(n)$*)*, the construction above still works, but now M' needs to systematically try (lexicographically) all possible 4-tuple locations of p, q, p, q. We need to assume that M always halts so that a simulation on a current 4-tuple that fails to accept can be abandoned by M' and proceed to the next lexicographic 4-tuple. This assumption can be made without loss of generality since any $S(n)$ space-bounded deterministic TM can be made halting [7]. □

Clearly, the first part of the proof above applies to unrestricted nondeterministic TMs (which are equivalent to deterministic TMs), i.e., to recursively enumerable sets:

Corollary 3 *The class of recursively enumerable sets is closed under* $dlad$.

One can obtain similar results for time-bounded TMs. For example, let *P* (*NP*) denote the class of languages accepted by polynomial time-bounded deterministic (nondeterministic) TMs . Then the constructions in the proof of Proposition 7 also proves:

Corollary 4 *P and NP are closed under* $dlad$.

Turning now to the operation of ld, we have:

Proposition 8 *DSPACE(*$S(n)$*), NSPACE(*$S(n)$*), P, and NP are not closed under* ld.

Proof. Let $L \subseteq \Sigma^*$ be a recursively enumerable language which is not in *DSPACE*$(S(n))$ (respectively, *NSPACE*$(S(n))$, *P*, *NP*). Let $a, b, \#$ be new symbols not in Σ.

One can easily show that there exists a language L' in *DSPACE*$(S(n))$(respectively, *NSPACE*$(S(n))$, *P*, *NP*) such that L' consists of words of the form $a^i b \# \alpha$ where $i \geq 0$ and $\alpha \in L$. Furthermore, for all $\alpha \in L$ there exists some $i \geq 0$ such that $a^i b \# \alpha \in L'$ (see, e.g., [19]). We now apply the ld operation:

$$ld(b \cdot L') \cap b\#\Sigma^* = b\#L$$

but $b\#L$ is not in *DSPACE*$(S(n))$ (respectively, *NSPACE*$(S(n))$, *P*, *NP*). □

However, for unrestricted TMs, it is easy to show:

Proposition 9 *The class of recursively enumerable sets is closed under* ld.

4. L-Systems

We now consider the closure properties of the families of 0L and ET0L languages under ld, $dlad$ and hi.

The closure properties of NCM and its generalizations and languages described by space- and time-bounded Turing machines for the hi operator have already been given in [2]. However, we wish to consider the closure properties of two families of L-systems here under all the operations of hi, ld and $dlad$. We will thus introduce the formal definition of the hi operation before we continue.

Definition 4 *Let* α *be a word in* Σ^+. *The hairpin inverse of* α, *denoted by* $hi(\alpha)$ *is defined as* $hi(\alpha) = \{xpy^r p^r z | \alpha = xpyp^r z$ *and* $x, y, z \in \Sigma^*, p \in \Sigma^+\}$.

This definition can be extended to languages in Σ^+ in the natural way.

A result similar to Lemma 1 and Lemma 2 showing that we can, without loss of generality, consider pointers of length one for the hi operation was shown in [2]:

Lemma 4 *If* $\alpha \in \Sigma^+$, $hi(\alpha) = \{xay^r az | \alpha = xayaz, a \in \Sigma, x, y, z \in \Sigma^*\}$.

We now show that the family of 0L languages is not closed under any of the considered operations.

Proposition 10 *The family of* 0L *languages is not closed under* hi, ld *or* $dlad$.

Proof. Consider $L = \{a\} \in$ 0L. Then

$$hi(L) = dlad(L) = ld(L) = \emptyset$$

which is not a 0L language. □

We now consider the closure properties of the family of ET0L languages.

Proposition 11 ET0L *is closed under* ld.

Proof. Follows immediately from Proposition 2 as ET0L is a full trio. □

The proofs for closure of ET0L under hi and $dlad$ are much more involved and are demonstrated below. We use some definitions used to describe synchronized context-free grammars, introduced by Jürgensen and Salomaa in [9]. We refer the reader to this paper for more intuition and examples.

A *tree domain* D is a nonempty finite subset of $\mathbb{N}^*$ such that

(i) If $\mu \in D$, then every prefix of μ belongs to D,

(ii) For every $\mu \in D$ there exists $i \geq 0$ such that $\mu j \in D$ if and only if $1 \leq j \leq i$.

Let A be a set. An *A-labelled tree* is a mapping $t : D \to A$, where D is a tree domain. Elements of D are called nodes of the tree and D is said to be the domain of t, $\mathrm{dom}(t)$. A node $\mu \in \mathrm{dom}(t)$ is labelled by $t(\mu)$. The set of leaves of t is denoted $\mathrm{leaf}(t)$. The subtree of t at node μ is t/μ. When there is no confusion, we refer to a node simply by its label.

Nodes of a tree t that are not leaves are called *inner nodes* of t. The *inner tree of* t, $\mathrm{inner}(t)$, is the tree obtained from t by cutting off all the leaves. The *yield* of an A-labelled tree t, $\mathrm{yd}(t)$, is the word obtained by concatenating the labels of the leaves of t from left to right; the leaves being ordered by the lexicographic ordering of $\mathbb{N}^*$. For $\mu \in \mathrm{dom}(t)$, $\mathrm{path}_t(\mu)$ is the sequence of symbols of A occurring on the path from the root of t to the node μ. For $\mu_1, \mu_2 \in \mathrm{leaf}(t)$ with $\mu_1 \neq \mu_2$, the *join* of μ_1, μ_2, denoted $\mathrm{join}(\mu_1, \mu_2)$ is the longest word μ such that μ is a prefix of both μ_1 and μ_2. In other words, the join of two leaves is the unique node in which the two paths split into two distinct paths.

Let $G = (N, T, I, P)$ be a CF grammar. A $(N \cup T \cup \{\lambda\})$-labelled tree t is a *derivation tree of* G if it satisfies the following conditions:

(i) The root of t is labelled by the initial nonterminal, that is, $t(\lambda) = I$.

(ii) The leaves of t are labelled by terminals or by the symbol λ.

(iii) Let $\mu \in \mathrm{dom}(t)$ have k immediate successors, $k \geq 1$. Then $t(\mu) \to t(\mu 1) \dots t(\mu k) \in P$.

A tree t is a partial derivation tree if (i) and (iii) are satisfied. The set of derivation trees of G is denoted $T(G)$. The derivation trees of G are in one-to-one correspondence with the equivalence classes of derivations of G producing terminal words, and thus

$$L(G) = \{\mathrm{yd}(t) \mid t \in T(G)\}.$$

Definition 5 *A* synchronized context-free grammar, SCF *grammar, is a context-free grammar*

$$G = (N, T, I, P), \tag{1}$$

where $N = V \times (S \cup \{\lambda\})$ for finite alphabets V and S.

We call elements of V the base nonterminals *and elements of S the* situation symbols. *Nonterminals of $V \times S$ are called the* synchronizing nonterminals *of G. Nonterminals of the form $V \times \{\lambda\}$, are called* nonsynchronizing nonterminals.

We define the morphism $h_G : (V \times (S \cup \{\lambda\}))^* \longrightarrow S^*$ by the condition $h_G((v, x)) = x$ for all $v \in V$ and $x \in S \cup \{\lambda\}$.

Definition 6 *Let G be an* SCF *grammar and $t \in T(G)$. Let $t_1 = \text{inner}(t)$ and let $\mu \in \text{leaf}(t_1)$. The* synchronizing sequence *(sync-sequence) corresponding to μ is* $\text{seq}_{t_1}(\mu) = h_G(\text{path}_{t_1}(\mu))$.

Definition 7 *Let $G = (N, T, I, P)$ be a* SCF *grammar where $N = V \times (S \cup \{\lambda\})$. A derivation tree $t \in T(G)$ is said to be* e*-acceptable if for all $\mu, \nu \in \text{leaf}(\text{inner}(t))$, $\text{seq}_{\text{inner}(t)}(\mu) = \text{seq}_{\text{inner}(t)}(\nu)$.*

Let G be a SCF grammar. The set of e-acceptable trees of G is denoted by $T_{\text{e}}(G)$.

Definition 8 *The language* e-synchronized generated *by G is $L_{\text{e}}(G) = \text{yd}(T_{\text{e}}(G))$. In this case, we call G an* e*-SCF grammar. A language L is an* e-SCF language*, if there exists a* SCF *grammar G such that $L = L_{\text{e}}(G)$.*

The family of e-SCF *languages is denoted by $\mathcal{L}_{\text{e}}(\text{SCF})$.*

Definition 9 *Let G be an* e-SCF *grammar. Then we say G is in binary normal form if all productions are of the form either $(A_x, x) \to (C_y, y)(D_y, y)$ or $(A_x, x) \to a,\ x, y \in S, a \in T \cup \{\lambda\}$*

It is known that we can assume without loss of generality that every e-SCF language is generated by an e-SCF grammar in binary normal form [1, 14]. Furthermore, it is known that the family of e-SCF languages is equal to the family of ET0L languages [13].

Intuitively, a derivation tree without nonsynchronizing nonterminals is e-acceptable if at each height, every nonterminal has the identical situation symbol. The use of nonsynchronizing nonterminals allows for nonterminals that need not synchronize with nonterminals in other branches. However, every tree in binary normal form does not have nonsynchronizing nonterminals. The language e-synchronized generated by G is the language obtained from the yields of only the e-acceptable derivation trees.

We let V^z be the set of all elements of V with z as superscript (not exponentiation), where z is a new symbol. Also, we say $v^\lambda = v$.

Proposition 12 *$\mathcal{L}_{\text{e}}(\text{SCF})$ is closed under hi.*

Proof. Let $G = (N, T, I, P)$ be an SCF grammar where $N = V \times (S \cup \{\lambda\})$ is in binary normal form. We will first transform G into an intermediate SCF grammar $G_1 = (N_1, T, I', P_1)$ with $N_1 = V_1 \times (S \cup \{\lambda\}), V_1 = V \cup V^{La} \cup V^{Ra} \cup V^{LaRa} \cup \{I'\}$ for all $a \in T$ and let P_1 be defined as P in addition to the following productions:

Let $I' \to I^{LaRa},\ \forall a \in T$.
For all productions in P of the form $(A_x, x) \to (B_y, y)(C_y, y)$, add
$(A_x^{LaRa}, x) \to (B_y^{LaRa}, y)(C_y, y) \mid (B_y, y)(C_y^{LaRa}, y) \mid (B_y^{La}, y)(C_y^{Ra}, y)$,
$(A_x^z, x) \to (B_y, y)(C_y^z, y) \mid (B_y^z, y)(C_y, y),\ \forall z \in \{La, Ra\}, a \in T$
For all productions in P of the form $(A_x, x) \to a, a \in T$, add
$(A_x^z, x) \to a \in P'$, for $z \in \{La, Ra\}$.

It is clear[2] that $L_{\mathrm{e}}(G_1) = \{w \mid w \in L_{\mathrm{e}}(G), |w|_a \geq 2, \text{ for some } a \in T\}$. Indeed, G_1 nondeterministically guesses some terminal, a say, and two paths, only resulting in a derivation tree if the two paths reach two distinct occurrences of the terminal a. Thus, G_1 has the property that for each e-acceptable derivation tree, each base nonterminal of the left and right path are marked with La and Ra, respectively, from the root to two distinct leaves, which are labelled by the same terminal, a. Moreover, for each tree t, the two paths separate at some node which is the unique join which we will denote by μ_t.

We would like to easily "turn the subtree around" in between the two marked paths, in order to simulate the effect of the hi operator. It is easy to reverse an entire subtree (as we will see in the transformation from G_2 to $\bar{G}$), however there may be branches on the subtree t/μ_t that are either to the left of the left marked path or to the right of the right marked path (see the first diagram of Figure 1). So, we cannot simply reverse the subtree t/μ_t.

Consequently, we will next move these branches outside of the two paths using nondeterminism.

Next, we transform G_1 into $G_2 = (N_2, T, I', P_2)$, another intermediate e-SCF grammar (see Figure 1 for intuition with this step of the construction), where $N_2 = V_2 \times (S_2 \cup \{\lambda\}), V_2 = V \cup V^{La} \cup V^{Ra} \cup V^{LaRa} \cup V^{'Ra} \cup \{I', L, R\}$ and $S_2 = S \cup V^L \cup V^R$, for all $a \in T$. P_2 has all the productions of P_1, however for all $a \in T$, make the following changes:

change productions of the form $(A_x^{La\,Ra}, x) \to (B_y^{La}, y)(C_y^{Ra}, y)$ to,
(1) $(A_x^{La\,Ra}, x) \to (L, y)(B_y^{La}, y)(C_y^{Ra}, y)(R, y)$,
change productions of the form $(A_x^{La}, x) \to (B_y, y)(C_y^{La}, y)$ to,
(2) $(A_x^{La}, x) \to (C_y^{La}, B_y^{L})$ and
$(C_y^{La}, B_y^{L}) \to (C_y^{La}, y) \mid (C_y^{La}, v), \forall v \in V^R$,
change productions of the form $(A_x^{Ra}, x) \to (B_y^{Ra}, y)(C_y, y)$ to,
$(A_x^{Ra}, x) \to (B_y^{Ra}, C_y^{R}) \mid (B_y^{'Ra}, v)$,
$(B_y^{'Ra}, v) \to (B_y^{Ra}, C_y^{R})$,
$(B_y^{Ra}, C_y^{R}) \to (B_y^{Ra}, y), \forall v \in V^L$.
For all productions of the form $(A_x, x) \to (B_y, y)(C_y, y)$, add
(3) $(A_x, x) \to (B_y, v)(C_y, v),\ v \in V^L \cup V^R$.
For all productions of the form $(A_x^{La}, x) \to (B_y^{La}, y)(C_y, y)$, add
$(A_x^{La}, x) \to (B_y^{La}, v)(C_y, v),\ v \in V^R$.
For all productions of the form $(A_x^{Ra}, x) \to (B_y, y)(C_y^{Ra}, y)$, add
(4) $(A_x^{Ra}, x) \to (B_y, v)(C_y^{Ra}, v),\ v \in V^L$.
In addition, add:
$(A_x^{La}, v) \to (A_x^{La}, x), \forall A_x^{La} \in V^{La}, v \in V^R$,
$(A_x^{Ra}, v) \to (A_x^{Ra}, x), \forall A_x^{Ra} \in V^{Ra}, v \in V^L$,
$(A_x, v) \to (A_x, u), \forall A_x \in V, v \in V^L \cup V^R, u \in V^L \cup V^R \cup \{x\}$,
$(R, r) \to (R, s) \mid \lambda, \forall r, s \in S \cup V^L$,
$(R, s) \to (R, A_x^R)(A_x, A_x^R), \forall A_x \in V, s \in S \cup V^L$,
$(R, A_x^R) \to (R, s), \forall A_x \in V, s \in S$,
$(L, r) \to (L, s) \mid \lambda, \forall r, s \in S \cup V^R$,
(5) $(L, s) \to (A_x, A_x^L)(L, A_x^L), \forall A_x \in V, s \in S$,
$(L, A_x^R) \to (L, s), \forall A_x \in V, \forall s \in S \cup V^R$.

[2] $|w|_a$ is equal to the number of occurrences of the letter a in the word w and $|w|$ is the length of w.

Let $t \in T_{\mathrm{e}}(G_1)$. We construct an accepting tree $t' \in T_{\mathrm{e}}(G_2)$. We denote by $\mathrm{sen}_t(n)$, the word obtained by concatenating the labels of all branches of t at height n from left to right. We proceed by induction on the height of t. Assume that at height n of t, the corresponding partial derivation tree t' of height m constructed so far has $\mathrm{sen}_t(n) = h(\mathrm{sen}_{t'}(m))$ where h is a homomorphism that fixes all nonterminals except it erases those with base nonterminals of L and R. There are many cases for height $n+1$. If there is a letter of $\mathrm{sen}_t(n+1)$ with $LaRa$ as superscript, then t' is constructed identically. If $\mathrm{sen}_t(n+1)$ has two distinct labels $(C_y^{La}, y)(D_y^{Ra}, y)$, then t' is constructed identically, but rather with $(L, y)(C_y^{La}, y)(D_y^{Ra}, y)(R, y)$ as children of the $LaRa$ node. It is clear that $\mathrm{sen}_t(n+1) = h(\mathrm{sen}_{t'}(n+1))$. If at height $n+1$, there is a branch with La as superscript which is the left child of its parent and a branch with Ra as superscript which is the right child of its parent, then all nonterminals of t' can be rewritten as in t and $\mathrm{sen}_t(n+1) = h(\mathrm{sen}_{t'}(m+1))$. If at height n of t, there are two nonterminals (A_x^{La}, x) and (B_x^{Ra}, x) which have children $(C_y, y)(D_y^{La}, y)$ and $(E_y, y)(F_y^{Ra}, y)$ respectively, then in t', (A_x^{La}, x) has one child (D_y^{La}, C_y^L) which has one child (D_y^{La}, y). The nonterminal (L, x) must consequently have children $(C_y, C_y^L)(L, C_y^L)$ which have children (C_y, y) and (L, y) respectively. Since the path with every base nonterminal of L is always directly to the left of the path with La as superscript, and all other nonterminals have identical children at height $n+1$ of t as in height $m+2$ of t', $\mathrm{sen}_t(n+1) = h(\mathrm{sen}_{t'}(m+2))$. The process is symmetric when the left path branches left and the right branches left. If the left path branches right and the right path branches left, we use the left symbol first, while the right path waits for the next height. Thus, by induction, and since L, R paths end with λ, $yd(t) = yd(t')$ and $L_{\mathrm{e}}(G_1) \subseteq L_{\mathrm{e}}(G_2)$.

For the reverse inclusion, we notice that at heights before the join, the tree must rewrite using only situation symbols from S since the paths must. After the join, whenever the left path branches left and the right branch branches right, all nonterminals must use situation symbols from S since the paths do (for an e-acceptable tree). Whenever the left path branches right and the right path branches left, the tree must rewrite using symbols from V^L, V^R then S since the paths do. Similarly for all other cases. Consequently, there is exactly one tree in $T_{\mathrm{e}}(G_2)$ for every tree in $T_{\mathrm{e}}(G_1)$. Thus, $L_{\mathrm{e}}(G_2) \subseteq L_{\mathrm{e}}(G_1)$.

Hence, we have $L_{\mathrm{e}}(G_1) = L_{\mathrm{e}}(G_2)$. Moreover, G_2 has the property that every nonterminal with La as superscript is the left child or only child of its parent and similarly for the right path.

Lastly, we transform G_2 into $\bar{G} = (\bar{N}, T, I', \bar{P})$ where $\bar{N} = \bar{V} \times (S_2 \cup \{\lambda\})$. For all the productions that can be performed inside the two paths, we add a new production with a marker $^-$ on top of each base nonterminal, with all the the productions reversed (ie. if original production is $A \to BC$, add $\bar{A} \to \bar{C}\bar{B}$). Then, for the productions used at the join, we switch the two nonterminals between the L and R path and add a $^-$ as superscript, leaving all productions outside the paths using normal productions. Hence, $L_{\mathrm{e}}(\bar{G}) = hi(L_{\mathrm{e}}(G))$. □

Corollary 5 ET0L *is closed under* hi.

Intuitively, with $dlad$, we pick four paths, and switch the yield between the first two with the yield between the second two. Similarly to the proof above, we will first pick two paths and manipulate the trees so that each base nonterminals labelled with La (or Ra, respectively) is the left child (or right) of its parent, then, we will add in the second set of paths to the right

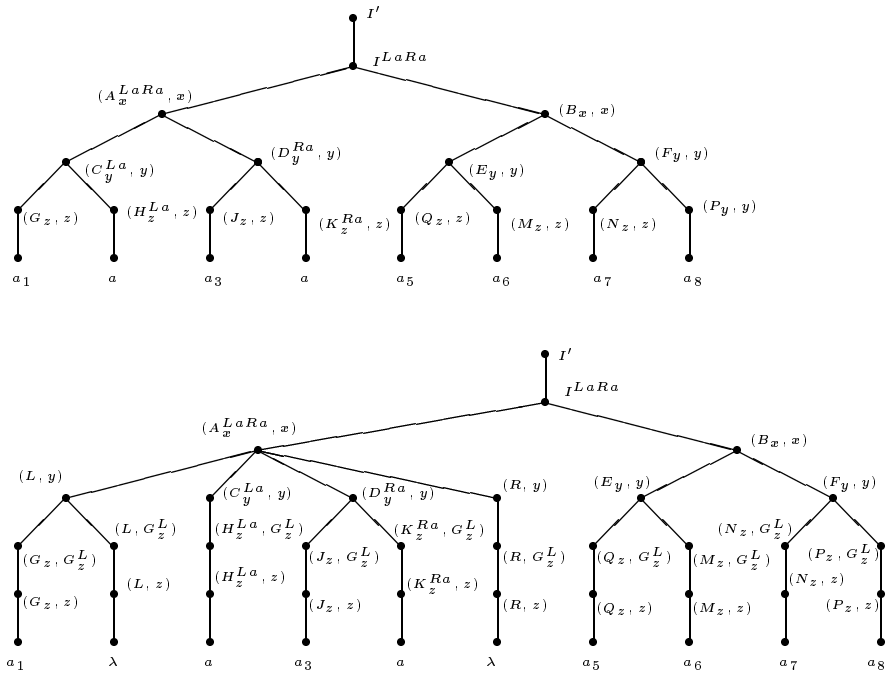

Figure 1: A derivation tree $t \in T_{\mathrm{e}}(G_1)$ and the new derivation tree $t' \in T_{\mathrm{e}}(G_2)$ corresponding to t.

of the first set and manipulate this subtree so as not to interfere with the first. Indeed, we need only switch these two subtrees to obtain the desired effect. However, the two joins may be at different heights, so we must guess at the lower join what the label will be at the higher join.

Proposition 13 *$\mathcal{L}_{\mathrm{e}}(\mathrm{SCF})$ is closed under $dlad$.*

Proof. Let $G = (N, T, I, P)$ be a e-SCF grammar in binary normal form where $N = V \times (S \cup \{\lambda\})$. Similarly to the construction for the hi operator, we will transform G into G_1 and then G_2 by first picking two paths from root to leaves and then moving all the child branches of the paths that are outside of the two paths to the L and R branches. The only alteration we must make to the construction is to allow separate terminals to be picked as pointers. This can easily be achieved by adding all productions of the form $I \to I^{LaRb}$ for all $a, b \in T$. Also, instead of using L and R on superscripts and nonterminals, we replace everywhere with L_1 and R_1. This is mainly to differentiate between the first set of two paths and the second set of two paths soon to be added. Hence $L_{\mathrm{e}}(G_2) = \{w \mid w \in L_{\mathrm{e}}(G), |w| \geq 2\}$.

Next, we wish to add a second set of paths to each accepting tree of G_2. However, the new paths can not interfere with the altered subtree of the first two. This is not difficult, as we only need to restrict the possible placements of the second two paths in the conversion between G and G_1.

First, for simplicity, we construct G_3 by changing all productions of the form $(A, x) \to (B, y)$ to $(A, x) \to (B, y)(\&, y)$ for all productions with one nonterminal of the right

hand side in P_2 and adding $(\&, u) \to (\&, v)(\&, v) \mid \lambda, \forall u, v \in S_2$. This does not change the accepted language and all productions either have one terminal, the empty word, two nonterminals or four nonterminals on the right hand side. Then, we transform, G_3 into G_4 by changing all base nonterminals into ordered pairs, keeping track of the old base nonterminal in the first coordinate and the situation symbol in the second. We will also use square brackets for clarity (ie. replace all nonterminals (A, x) in every production of G_3 with $([A, x], x)$). This is since the conversion from G_2 to G_3 creates trees that no longer keep track of the situation symbol on the subscript of the base nonterminal.

Next, we transform into G_5 by adding the possible second set of paths to each accepting tree of G_4 (similarly to the conversion of G to G_1). We will restrict that the second set always be to the right of the first two. This is achieved as follows:

For all productions of form $I' \to I^{L_1aR_1b},\ a, b \in T$, change to
$I' \to I^{L_1aR_1bL_2aR_2b}$ and add $([A^z, x], x) \to a, \forall x \in S, z \in \{L_2a, R_2a\}$.
For all productions of form $([A^{L_1aR_1b}, x], x) \to ([B, y], y)([C^{L_1aR_1b}, y], y)$, add
$([A^z, x], x) \to ([B, y], y)([C^z, y], y), \forall z \in \{L_1aR_1bL_2a, L_1aR_1bL_2aR_2b\}$.
For all productions of form $([A^{L_1aR_1b}, x], x) \to ([B^{L_1aR_1b}, y], y)([C, y], y)$, add
$([A^z, x], x) \to ([B^z, y], y)([C, y], y), \forall z \in \{L_1aR_1bL_2a, L_1aR_1bL_2aR_2b\}$ and
$([A^{L_1aR_1bL_2aR_2b}, x], x) \to ([B^{L_1aR_1bL_2a}, y], y)([C^{R_2b}, y], y) \mid$
$([B^{L_1aR_1b}, y], y)([C^{L_2aR_2b}, y], y)$.
For all productions of form $([A, x], x) \to ([B, y], y)([C, y], y)$, add
$([A^z, x], x) \to ([B^z, y], y)([C, y], y) \mid ([B, y], y)([C^z, y], y)$,
$\forall z \in \{L_2a, R_2b, L_2aR_2b\}, \forall a, b \in T$,
$([A^{L_2aR_2b}, x], x) \to ([B^{L_2a}, y], y)([C^{R_2b}, y], y), \forall a, b \in T$.
For all productions of form
$([A, x], x) \to ([L_1, y], y)([B, y], y)([C, y], y)([R_1, y], y)$, add
$([A^z, x], x) \to ([L_1, y], y)([B, y], y)([C, y], y)([R_1^z, y], y)$,
for $z \in \{L_2a, L_2aR_2b\}, \forall a, b \in T$.

Thus,

$$L_{\mathrm{e}}(G_5) = \{w \mid w \in L_{\mathrm{e}}(G), w = u_1au_2bu_3au_4bu_5, \text{ for some } a, b \in T, u_i \in T^*\}.$$

Moreover, there are third and fourth paths, both always being on the second path or to the right of it. They both must end strictly to the right of the second path, hence it can not continue on the second path after the join of the first two paths.

Now that we have the paths marked for each tree, we can do a similar conversion to G_6 as we did from G_1 to G_2 except with respect to the L_2a and R_2b paths instead of L_1a and L_2b (ie. ignoring L_1a and R_1a on base nonterminals) and using the second coordinate of the base nonterminal instead of the subscript. The only difference that must be considered are with productions of the form

$$([A^{L_1aR_1bL_2a}, x], x) \to ([L_1, y], y)([B^{L_1a}, y], y)([C^{R_1b}, y], y)([R_1^{L_2a}, y], y).$$

Here, the tree is not binary, and when the L_2 path branches right, it should use the left branch as situation symbol. Fortunately, nonsynchronizing nonterminals make it easy to do just that. Instead change the production above to

$$([A^{L_1aR_1bL_2a}, x], x) \to ([L_1, y][B^{L_1a}, y][C^{R_1b}, y], \lambda)([R_1^{L_2a}, y], y)$$

and add

$$([L_1, y][B^{L_1 a}, y][C^{R_1 b}, y], \lambda) \to ([L_1, y], y)([B^{L_1 a}, y], y)([C^{R_1 b}, y], y)$$

in advance. Then, when converting, it will pass this new base nonterminal as situation symbol and the L_2 branch will continue by duplicating the original subtree between the first two paths. Otherwise, we treat other productions with four nonterminals on the right hand side exactly as we treat two. Since the conversion between G_1 and G_2 will work for arbitrary grammars with paths marked as in G_1, and G_5 is of this form except productions with four nonterminals on the right hand side which we treat similarly, we obtain $L_{\mathrm{e}}(G_6) = L_{\mathrm{e}}(G_3)$.

Finally, it suffices to "switch" the two subtrees below the joins. Notice now that the first two marked paths of G_6 no longer have to be in continuous paths from root to leaves, as they can be passed at their join to the L_2 branch. However, the two subtrees are completely disjoint, not sharing any nodes. So, we can talk about *the* left and right subtree since neither has the other in its subtree.

Currently, when the joins occur, there are either three or four nonterminals on the right hand side, with only the second and third being between the paths. We wish to switch these two to the other join. This is easier to do if we need only pass one nonterminal. So, we transform into G_7 by changing the second and third nonterminal in these productions to an intermediate nonsynchronizing nonterminal which then immediately gets rewritten to the original two. We will also mark these nonsynchronizing nonterminals with J_1 or J_2 on the superscript of the base nonterminal if they are the left or right join, respectively. Then, we convert G_7 into G_8 by changing it into binary normal form. This conversion introduces a new situation symbol, \$, which will be used in place of λ as situation symbol and will preserve the yield of the subtree at each join. We convert into G_9 by re-marking the two paths until the two joins on the superscript by 1 and 2, only accepting if each reaches a symbol from $V_8^{J_1} \times \{\$\}$ and $V_8^{J_2} \times \{\$\}$, respectively. We need to keep track of paths until join which were not preserved by previous constructions.

We construct $G_{10} = (N_{10}, T, I', P_{10})$ where $N_{10} = V_{10} \times (S_{10} \cup \{\lambda\}$ by switching subtrees as follows:

For all productions of form $(A_x^i, x) \to (B_{\$}^{J_i}, \$)(C_{\$}, \$)$, for $i = 1, 2$, change to
(1) $(A_x^i, x) \to (J_i, B_{\$}^{J_i})(C_{\$}, B_{\$}^{J_i})$,
(2) $(\bar{A}_x^i, x) \to (\#, B_{\$}^{J_i} C_{\$})$,
(3) $(A_x^i, x) \to (D_{\$}, B_{\$}^{J_1} D_{\$}^{J_2})(C_{\$}, B_{\$}^{J_1} D_{\$}^{J_2})$, $\forall D_{\$} \in V_9$, if $i = 1$,
(4) $(A_x^i, x) \to (D_{\$}, D_{\$}^{J_1} B_{\$}^{J_2})(C_{\$}, D_{\$}^{J_1} B_{\$}^{J_2})$, $\forall D_{\$} \in V_9$, if $i = 2$.
If J_i is instead superscript on the base nonterminal $C_{\$}$,
(1), (3) and (4) are completely analogous, however we change (2) to
(5) $(\bar{A}_x^i, x) \to (B_{\$}, C_{\$}^{J_i})(\#, C_{\$}^{J_i})$.
For all productions of form $(A_x^i, x) \to (B_y^i, y)(C_y, y)$, add
(6) $(A_x^i, x) \to (\bar{B}_y^i, D_w^{J_k})(D_w, D_w^{J_k})(R_i, D_w^{J_k})(C_y, D_w^{J_k})$ $\forall D_w \in V_9, k \neq i$,
(7) $(\bar{A}_x^i, x) \to (\bar{B}_y^i, C_y^{R_i})$.
For all productions of form $(A_x^i, x) \to (B_y, y)(C_y^i, y)$, add
(8) $(A_x^i, x) \to (B_y, D_w^{J_k})(\bar{C}_y^i, D_w^{J_k})(D_w, D_w^{J_k})(R_i, D_w^{J_k})$ $\forall D_w \in V_9, k \neq i$,
(9) $(\bar{A}_x^i, x) \to (B_y, y)(\bar{C}_y^i, y)$.

For all productions of form $(A_x, x) \to (B_y, y)(C_y, y)$, add
(10) $(A_x, x) \to (B_y, u)(C_y, u)\ \forall u \in V_9^{J_i} \cup V_9^{R_i} \cup V_9^{J_1} V_9^{J_2} \cup V_9^{J_i} V_9,\ \forall i.$
In addition, add
$(A_x, u) \to (A_x, x),\ \forall u \in V_9^{J_i} \cup V_9^{R_i} \cup V_9^{J_1} V_9^{J_2} \cup V_9^{J_i} V_9,\ \forall i,$
$(\#, u) \to (\#, v) \mid \lambda,\ \forall u \in V_9^{J_i} \cup S_9,\ \forall v \in S_9,\ \forall i,$
(11) $(R_i, x) \to (R_i, A_y^{R_i})(A_x, A_y^{R_i}),\ \forall A_y \in V_9,\ \forall x \in S_9,\ \forall i,$
$(R_i, u) \to (R_i, v) \mid \lambda,\ \forall u \in S_9 \cup V_9^{J_i} \cup V_9^{J_k},\ \forall v \in S_9 \cup V_9^{J_k},\ \forall i, k \neq i,$
$(J_i, u) \to (J_i, v),\ \forall u \in V_9^{J_i} \cup V_9 \cup V_9^{R_i} \cup V_9^{J_i} V_9,\ \forall v \in V_9 \cup V_9^{R_i},\ \forall i,$
$(J_i, u) \to (A_\$, A_\$^{J_k}) \mid (A_\$, A_\$^{J_k} B_\$),\ \forall u \in S_9,\ \forall A_\$, B_\$ \in V_9, k \neq i,$
$(\bar{A}_x^i, u) \to (\bar{A}_x^i, x),\ \forall u \in V_9^{R_i} \cup V_9^{J_k}, k \neq i,$
$(R_i, x) \to (B_\$, A_\$^{J_i} B_\$),\ \forall A_\$, B_\$ \in V_9.$

Let $t \in T_{\mathrm{e}}(G_9)$. We will construct $t' \in T_{\mathrm{e}}(G_{10})$. Construct t' exactly as t until the first join. Indeed, this is the only way to construct t'. If both branches reach joins at the same height, the two paths use productions from (3) and (4) which will switch labels of the joins using the situation symbol. All other branches at this height simply use this same situation symbol using (10) and then continue as in t. The rest of the tree is identical except with the two subtrees switched. The only other ways to construct t' involve all branches using productions created in (10), however this will not change the yield. If the left path reaches the join at a higher height, then, at the join, it uses the production created in (1). The right path nondeterministically uses the same situation symbol using productions from either (6) or (8). The situation symbol allows the right path to continue the subtree at the left join. However, the yield of this subtree need be between the yield of all branches created on the right path. The nonterminal J_i nondeterministically uses the same situation sequence as all the other branches until the right path reaches the second join. Also, the right path continues the derivation using productions from (7) or (9) depending on whether the path branches left or right. If the path branches right, the passed subtree continues, however if it branches left, the right child should be continued to the right of the passed subtree. Consequently, it uses the production in (7) which is continued by R_i using the production in (11). Then, the nonterminal J_i continues the subtree at this join. The right path continues nondeterministically using situation symbols only from S_9. Similarly if the right join occurs first. Thus, $t' \in T_{\mathrm{e}}(G_{10})$ and this is the only way to construct an accepting tree.

Hence, $L_{\mathrm{e}}(G_{10}) = \{w \mid w \in dlad(L_{\mathrm{e}}(G))\}$. □

Corollary 6 ET0L *is closed under* $dlad$.

5. Language Equations

We now address the solvability of equations of the form $ld(X) = R$, $dlad(X) = R$ where R is a given language and X an unknown.

We are able to show that the question of the existence of a solution for equations where R is regular is decidable, using essentially the same technique as for equations involving the hi operation, [2]. In fact, we are able to generalize our results to language equations involving unary operators as follows. Consider language equations of the type $op(X) = R$,

where $op : \Sigma^+ \longrightarrow 2^{\Sigma^+}$ is a unary word operation generalized to a language operation in the natural way:

$$\text{op}(L) = \bigcup_{\alpha \in L} \text{op}(\alpha), \text{ for any } L \subseteq \Sigma^+.$$

To solve equations of the form $op(X) = R$, we need the notion of an inverse of the operation *op*, defined as follows.

Definition 10 *Let $op : \Sigma^+ \longrightarrow 2^{\Sigma^+}$ be a unary word operation. The inverse of the operation op, denoted by op^{-1}, is defined as the unary word operation with the property that, for all $u, v \in \Sigma^+$, $u \in op(v)$ iff $v \in op^{-1}(u)$.*

Note that the operation "is the inverse of" is symmetric.

Lemma 5 *The inverses of the operations $hi, ld, dlad$, reversal are respectively hi, li, $dlad$ and reversal where li is the operation defined as $li(u) = \{xayaz |\ u = xaz, x, y, z \in \Sigma^*, a \in \Sigma.\}$*

The existence of an operation inverse to *op* allows us to address equations $op(X) = R$, R a given language and X the unknown, as follows.

Proposition 14 *Let $R \subseteq \Sigma^*$ be a language. If the equation $op(X) = R$ has a solution L, then the language $X_{max} = [op^{-1}(R^c)]^c$ is a maximal solution.*

Proof. Claim 1: $op(X_{\max}) \subseteq R$. Proof by contradiction. Suppose there exists $u \in op(X_{\max})$ such that $u \notin R$. Then clearly $u \in R^c$. As $u \in op(X_{\max})$, then $u \in op(v)$, where $v \in X_{\max}$. By the definition of the inverse of a unary operation, this implies that $v \in op^{-1}(u) \subseteq op^{-1}(R^c)$. This is a contradiction, since $v \in X_{\max} = [op^{-1}(R^c)]^c$.

Claim 2: If $L \subseteq \Sigma^+$ is a language such that $op(L) \subseteq R$, then $L \subseteq X_{\max}$. We use again a proof by contradiction. Suppose there exists $L \subseteq \Sigma^+$ such that $op(L) \subseteq R$ and $L \not\subseteq X_{\max}$. Then there must exist $u \in L - X_{\max}$. As $u \notin X_{\max}$, $u \in op^{-1}(R^c)$ which implies that $u \in op^{-1}(v)$ with $v \in R^c$. However, since $u \in L$, we have that $v \in op(u) \subseteq op(L) \subseteq R$, a contradiction with the fact that $v \in R^c$.

Thus if the equation $op(X) = R$ has a solution L, then by Claim 2 $L \subseteq X_{\max}$. By Claim 1 we have that $R = op(L) \subseteq op(X_{\max}) \subseteq R$ and thus $op(X_{\max}) = R$. □

Proposition 15 *If $R \subseteq \Sigma^+$ is a regular language, the problem of whether of not the equation $dlad(X) = R$ (respectively $hi(X) = R$, ld(X) = R, $X^r = R$) has a solution $X \subseteq \Sigma^*$ is decidable.*

Proof. Construct $R' = [dlad(R^c)]^c$ (respectively $R' = [hi(R^c)]^c$, $R' = [li(R^c)]^c$, $R' = [(R^c)^r]^c$). If the equation $dlad(X) = R$ (respectively $hi(X) = R$, ld(X) = R, $X^r = R$) has a solution, then by Proposition 14, R' is also a solution. A decision algorithm would thus consist of constructing R' and checking that $dlad(R') = R$ (respectively $hi(R') = R$, $li(R') = R$, $R'^r = R$). Since the equality of regular languages is decidable, and REG is constructively closed under $dlad, hi, li$ and reversal the proposition follows. (REG is closed

under li as, for a language $L \subseteq \Sigma^+$ we have that $li(L) = s(g(L))$ where g is a gsm defined so that $g(xay) = xa'y$ for all $x, y \in \Sigma^*$, $a \in \Sigma$, while s is the regular substitution $s : \Sigma \cup \Sigma' \longrightarrow 2^{\Sigma^*}$ defined as $s(a) = a$ for $a \in \Sigma$ and $s(a') = a\Sigma^* a$.) □

For case where R is context-free, it turns out that the decidability of existence of solutions to equations $op(X) = R$ is obtained when op is an involution. An involution is a function $f : A \longrightarrow A$ with the property that $f(f(x)) = x$. For example, the identity and the reversal operators are involutions.

Proposition 16 *If $op : 2^{\Sigma^*} \longrightarrow 2^{\Sigma^*}$ is a unary language operator that is also an involution then the equation $op(X) = R$ always has a solution.*

Proof. The solution to the equation is $op(R)$. □

6. Conclusions

In this paper we have continued the work in [2, 3] and considered the properties of generalizations of the bio-operations proposed in [5, 4, 18]: ld, $dlad$ and hi. The language families NCM(k), NPCM(K) and NFCM(k) were found to be closed under the ld operation and, more generally, it was shown that any full trio is closed under the ld operation. The families NCM and NFCM were shown to be closed under $dlad$ while the families NPCM(0) (CF), NPCM and NFCM(0) were shown to not be closed.

With respect to language families accepted by time- and space-bounded Turing machines, we demonstrated that NSPACE($S(n)$), DSPACE($S(n)$), *P* and *NP* were closed under $dlad$ while the same families were not closed under ld.

We then showed that the family of 0L languages is not closed under hi, ld or $dlad$ operations while the family of ET0L languages is closed under all three operations.

Finally, we considered language equations over the hi, ld and $dlad$ operations, and showed that it is decidable whether or not a solution to the equation $hi(X) = R$ (respectively $ld(X) = R$, $dlad(X) = R$) exists when R is a regular language. We further showed that, in general, if op is a unary operation with an inverse op^{-1} then if the equation of the form $op(X) = R$ has a solution, $X = [op^{-1}(R^c)]^c$ is a maximal solution.

It is our hope that continued study of the abstract properties of these operations will contribute to a better understanding of the underlying biological processes upon which they are based. Future results on the biology of ciliates will allow us to validate our models, and conversely, theoretical results will provide insights into what the hard limits of the gene descrambling process are.

References

[1] H. Bordihn and M. Holzer. 2002. On the computational complexity of synchronized context-free languages. *J. of Universal Computer Science*, 8 (2): 119-140.

[2] M. Daley, O. H. Ibarra and L. Kari. 2002. Closure and decidability properties of some language classes with respect to ciliate bio-operations. *Theoretical Computer Science*, to appear.

[3] M. Daley and L. Kari. 2002. Some properties of ciliate bio-operations. *Preproceedings, Sixth International Conference on Developments in Language Theory*, 122-139.

[4] A. Ehrenfeucht, T. Harju, I. Petre and G. Rozenberg. 2001. Patterns of micronuclear genes in cilliates. *Lecture Notes in Computer Science*, 2340, 279-289.

[5] A. Ehrenfeucht, D.M. Prescott and G. Rozenberg. 2001. Computational aspects of gene (un)scrambling in ciliates. In *Evolution as Computation* (L.F. Landweber, E. Winfree eds.) Springer-Verlag, Berlin, Heidelberg, 45-86.

[6] T. Harju, O. H. Ibarra, J. Karhumäki and A. Salomaa. 2002. Some decision problems concerning semilinearity and commutation. *J. of Computer and System Sciences*, 65 (2): 169-439; extended abstract has appeared in *Lecture Notes in Computer Science*, 2076: 579-590, 2001.

[7] J. E. Hopcroft, R. Motwani and J. D. Ullman. 2001. "Introduction to Automata Theory, Languages, and Computation," *Addison Wesley*.

[8] O. H. Ibarra. 1978. Reversal-bounded multicounter machines and their decision problems. *J. of the Association for Computing Machinery*, 25: 116-133.

[9] H. Jürgensen and K. Salomaa. 1997. Block-synchronization context-free grammars. In *Advances in Algorithms, Languages, and Complexity* (Z. Du, I. Ko eds.), Kluwer Academic Publishers, The Netherlands, 111-137.

[10] L. Kari and L.F. Landweber. 2000. Computational power of gene rearrangment In *DNA5, DIMACS series in Discrete Mathematics and Theoretical Computer Science* (E. Winfree, D. Gifford eds.), American Mathematical Society, 54: 207-216.

[11] L.F. Landweber and L. Kari. 1999. The evolution of cellular computing: nature's solutions to a computational problem. *DNA4, BioSystems* (L. Kari, H. Rubin, D.H. Wood eds.), Elsevier, 52(1-3):3-13.

[12] L.F. Landweber, T. Kuo and E. Curtis. 2000. Evolution and assembly of an extremely scrambled gene. *Proc. Nat. Acad. Sci.*, 97(7): 3298-3303.

[13] I. McQuillan. 2002. The generative capacity of block-synchronized context-free grammars. *submitted*.

[14] I. McQuillan. 2002. Descriptional complexity of block-synchronization context-free grammars. In *Descriptional Complexity of Formal Systems, Pre-Proceedings of a workshop*, The University of Western Ontario, London Canada, Report No. 586.

[15] D.M. Prescott. 1992. Cutting, splicing, reordering, and elimination of DNA sequences in hypotrichous ciliates. *BioEssays*, 14(5): 317-324.

[16] D.M. Prescott. 1992. The unusual organization and processing of genomic DNA in Hypotrichous ciliates. *Trends in Genet.*, 8:439-445.

[17] D.M. Prescott. 2000. Genome gymnastics: unique modes of DNA evolution and processing in ciliates. *Nature Reviews Genetics*, 1:191-198.

[18] D.M. Prescott, A. Ehrenfeucht and G. Rozenberg. 2001. Molecular operations for DNA processing in hypotrichous ciliates. To appear in *European Journal of Protistology*.

[19] A.Salomaa, *Formal languages*, Academic Press, New York, 1973.